John von Neumann Institute for Computing


# Load Balancing Support for Grid-enabled Applications

S. Rips

published in

*Parallel Computing:*
*Current & Future Issues of High-End Computing*,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 97-104, 2006.



http://www.fz-juelich.de/nic-series/volume33

# Load Balancing Support for Grid-enabled Applications

S. Rips[a]

[a]Heinz Nixdorf Institute, University of Paderborn, 33102 Paderborn, Germany

Executing parallel applications in Grid environments, often leads to poor efficiency values due to different compute power and different networks. Hence, new strategies are required for reducing runtime. Especially, load balancing must be adapted to the characteristics of this heterogeneous environment. We developed a supporting module, which explores the used environment and supplies the load balancer with information. The load balancer considers this additional information while partitioning the data and operations. This leads to execution patterns with minimised communication and reduced idle times caused by blocking sends/receives.

## 1. Introduction

Grid Computing enables a seamless access to a large number of compute, storage, network, and other resources, which are used by different communities from industry and academia for solving complex problems. Aggregation of multi-site resources allows the processing of large-scale problems that are too complex for traditional clusters and other parallel machines [6]. However, despite the tremendous research efforts in developing Grid middleware and architecture components, a lack of running Grid-enabled applications is still noticeable. Difficult installation of Grid middleware and resource discovery is one of the major problems. Moreover, users are forced to modify and to adapt their applications to the heterogeneous Grid world, which significantly differs from the traditional cluster environment built around homogeneous processing elements and networks. A direct execution of the existing parallel applications in Grid environments often leads to poor efficiency values, as the differences between the involved processing elements and the long communication times due to the slow networks between Grid sites are not sufficiently considered. In order to solve this problem, decisive system software components such as the load balancer have to be aware of the characteristics of the current Grid environment. Therefore, a supporting module for any kind of load balancers has been developed, which explores the current environment and supplies the load balancer with this knowledge. The load balancer considers this additional information while partitioning the data and operations and thus leads to execution patterns with minimised intra-partition communication and reduced idle times due to blocking sends/receives. After a certain runtime the overhead created by the environment analysis is neglected as compared to the speed-up and efficiency values achieved with the optimised partitioning and distribution.

This paper presents the core method for monitoring and discovering the network topology and the characteristics of the involved compute sites. Subsequently, the developed architecture and the current implementation of the prototype are described. Finally, a set of performance measurements shows the quality of the developed solution by considering CFD (Computational Fluid Dynamics) applications as an example.

## 2. Related Work

The structure of the Grid comprises characteristics of homogeneous as well as heterogeneous systems, loosely coupled as well as tightly coupled systems.

Load balancing strategies aim to adapt the load optimally to the environment. However, they mainly consider the application running on a parallel, homogeneous system. Only a few methods address also the special characteristics of the underlying system.

Zaki et al. [10] consider different processor speeds and distribute the load adequately. However, processor speeds are obtained by a profiling run and they assume full connectivity among the processors, with uniform latency and bandwidth.

Hendrickson and Devine [4] review the major classes of dynamic load balancing (DLB) approaches. They point out that for heterogeneous systems, different amounts of computing power and memory should be considered. Additionally, they emphasise that network connections with different speeds are important for a DLB strategy. However, a solution is not proposed.

Kielmann et al. [5] emphasise that a collection of clusters can be seen as a hierarchical system. They use a tree topology to do load balancing for divide-and-conquer applications. However, they do not take different PE characteristics into account.

Willebeek and Reeves present in [9] a hierarchical balancing method (HBM). HBM is an asynchronous, global approach which organises the system into a hierarchy of subsystems. The strategy has been implemented on a hypercube system. Due to its hierarchical structure, it is not necessary to perform any analysis of the topology at the beginning as well as modifications during runtime.

Especially [5] and [9] show the importance of considering the underlying network.

We did not found any methods in the literature that detect the hierarchical structure of a Grid environment and use the gained results to optimise middleware, as e.g. load balancing, specifically for this structure.

The next section describes the analysis of the system in order to build a base for load balancing decisions. The first step is the analysis of the underlying network that results in a hierarchical subsystem, where each subsystem indicates small communication times inside and slower ones outside. The following discovery of each node's capacities and therewith the capacities of each subsystem is a further step to support load adaptation to a heterogeneous system.

The resulting structure constitutes an appropriate basis for the detection of load imbalance and for minimising the amount of PEs that participate on load balancing.

## 3. Grid Environment Discovery

### 3.1. Network

Based on monitoring the network, the system is organised into a hierarchy of subsystems that reflects the current system status independent of physical connections. Each level of the hierarchy represents a magnitude of communication speed.

Our method to create this hierarchical representation follows a distributed approach without having a global decision instance. This enables scalability, which is particularly important when using thousands of processing elements (PEs) as in the Grid.

Communication times, resulting from measurements, are used to set up the hierarchy levels. For the lowest level, each $PE_k$ builds a basic subsystem $Sub_k$ out of all PEs $PE_j, 1 \leq j \leq n$ with communication times $t(k,j)$, with:
Let $T^k_{com} = (t(k,i_1),...,t(k,i_{n-1}))$, where $t(k,i_j) \leq t(k,i_{j+1})$ and $i_j \neq k$.
Then, a subsystem $Sub_k$ is defined as:
$Sub_k = \{PE_{i_1},...,PE_{i_l}, PE_{i_p} \in T^k_{com} | l = min\{1,...,n-1\}, s.t. \frac{t(k,i_l+1)}{t(k,i_l)} \geq threshold > 1\}$,
with $threshold$ set to any fixed value.

This means, a subsystem is built out of PEs with smallest communication times. The ratio of the

PE with the smallest communication time outside the subsystem and those with the highest time inside the subsystem, is greater than $threshold$. The threshold indicates a jump in the list of sorted communication times, i.e. a next level of communication times.

By this procedure, magnitudes of communication speeds are considered. By setting the threshold to a small value ($1 < threshold < 1.8$), the resulting granularity of the hierarchy is much finer than setting this parameter to a high one.

Specific PEs are designated to control the hierarchy discovery of the next level. These PEs (master nodes) again combine subsystems to new ones based on their interconnection speeds as described for the lowest level. This proceeds until the whole system has been analysed.

During runtime, synchronisation points of the application are used to monitor the current network situation. If considerable changes are detected our system responses by adapting the subsystem hierarchy.

### 3.2. Compute Nodes

Besides this fundamental analysis of the network, support for Grid-enabled applications implies also the inspection of the PEs' capacities.

We fulfil this requirement by monitoring the processing speed of each PE while the application is running. This is done by measuring the time that is spent on calculating an application's load unit (e.g. a cell calculation in CFD applications). The gathered information is used to determine the optimal percentage of load each PE should work on. Based on this information, the imbalance for each subsystem is calculated.

## 4. Rebalancing

Rebalancing load between a few adjacent PEs can lead to a sufficient common load balance. Not all PEs must be involved in the load balancing procedure. Our module provides an appropriate subsystem of the hierarchy to the load balancing component of the application. All PEs belonging to a subsystem that is balanced in total but whose sub-subsystems are unbalanced, have to do load balancing.

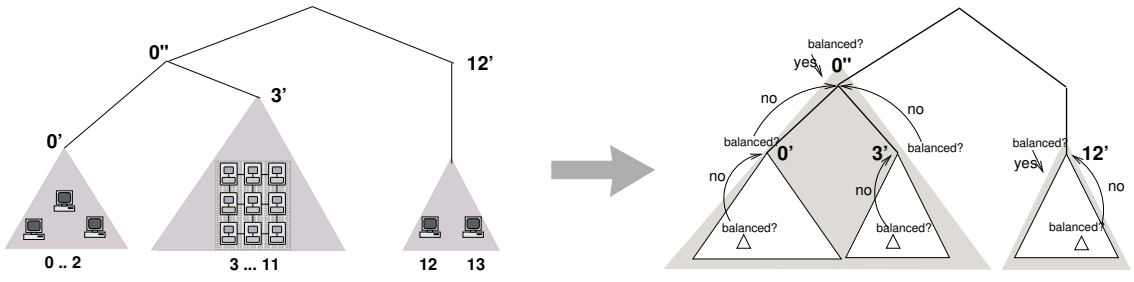


Figure 1. Subsystem hierarchy and detection of load imbalance

Figure 1 shows an example of detection of load balancing participants. Some PEs of the lowest level are over- or underloaded and send a request to their master nodes. The master nodes (here: 0', 3', 12') check whether the subsystem is balanced or not, i.e. whether the current load of the

subsystem differs from its optimal load. If it is balanced (here: subsystem 12') its associated nodes (here: PEs 12, 13) must do load balancing within the subsystem.

If the subsystem is not balanced, the request is passed to the master node of the next higher level (here: 0' and 3' pass requests to 0''). This proceeds until a balanced subsystem is found. Since the whole system is always balanced, this process stops at the latest at the highest level.

In the example, subsystem 0'' with PEs 0, ... 11 has to rebalance its load between its associated nodes but independently from subsystem 12'.

## 5. Architecture

Two goals determined the design of the architecture: transparency and minimising necessary changes of the application code. The last point led to the usage of MPI [2]. This decision was forced by the fact, that a lot of parallel codes of high performance applications as well as parallel load balancing tools use MPI to implement communication between processes. Furthermore, MPI provides a profiling interface. It enables the execution of extra code when calling designated MPI functions. This procedure is fully transparent for the MPI application.

The supporting module consists of two components (see figure 2). The communication monitor (CM) observes the network whereas the application monitor (AM) considers the capacity of the PEs.

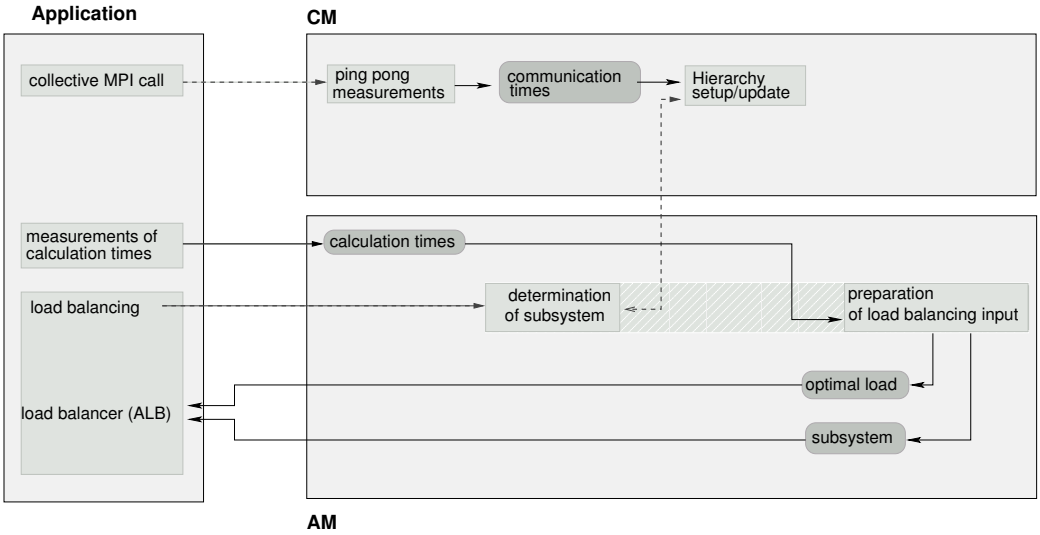


Figure 2. Architecture of Monitoring System

The CM uses the MPI profiling interface to make network monitoring transparent for the application. Thereby, no changes of the application code have to be done. It is sufficient to re-link the application to an extra library. The CM automatically sets up the subsystem hierarchy at program start based on measurements of ping-pong tests. These tests are done under the control of a pre-calculated communication schedule. It avoids conflicts in the send–receive procedure and minimises the number of ping-pong rounds.

The size of the messages is set to 1 kB. Tests with different message sizes (1 kB, 64 kB and 64 MB) have shown that the number of bytes sent is nonrelevant when detecting magnitudes of

communication times.

Whenever a collective MPI routine is called by the application, the CM performs communication measurements as done at the beginning. These measurements are used for subsystem updates, which are necessary before calling load balancing.
The application has access to the hierarchy via defined interfaces. The CM is realized as a library that provides a small API to the developer.

The integration of the AM requires only few modifications of the application code. Runtime measurements of a calculation unit must be done. The results must be passed together with load information via an API to the AM. These values are used by the AM to calculate the optimal load rate.

When load balancing is called by the application, the AM initiates the determination of the subsystem in which the rebalancing has to be done. First, it assigns the CM to check the hierarchy's up-to-dateness. If significant network changes occurred, the CM updates it. Based on this hierarchy, the AM then determines the PEs that participate on the same load balancing process as itself.

The preparation of load balancer input is the final task of the AM. Based on the calculation times provided by the application, together with the current load, the optimal load is calculated and passed to the application. This information is used as parameters for the application's load balancer (ALB).

The monitors on the different PEs have their own view on the system. Each CM/AM knows all members of its own subsystem and, in case of being a master node, keeps load information of its subsystem and the lower level subsystem. This information is sufficient when inspecting the balance from leaves to root. The absence of a global instance ensures scalability.

## 6. Prototype

For our prototype we use mesh based applications, e.g. CFD codes, that use domain decomposition for load balancing. In this area, several dynamic load balancing tools have been developed, but only few of them (e.g. Jostle [8], ParMetis [7]) can handle information about the required partition sizes, as provided by our software.

When load balancing is initiated, the subsystem hierarchy provided by the CM is extended by the load information delivered by the AM. An optimal load amount is determined and passed to the application's load balancer that calculates a new partitioning.

The determination of the participating PEs in load balancing is done by our module. It detects an appropriate subsystem and passes the corresponding PEs to the load balancer.

For Jostle, this is done by creating a new MPI communicator containing the current load balancing partners. Jostle performs the calculations of new partitions, with sizes given by the AM, on the PEs of this new communicator.

## 7. Performance evaluation

Runtime measurements were done on a PC-Cluster with InfiniBand [1] network and two processors (64-bit Intel Xeon) on each node, communicating via shared memory. Our algorithm is able to detect this two level hierarchy (shared memory and InfiniBand communication).

Tests with PACX-MPI[3] had shown that further hierarchy levels are detected by our algorithm. With support of PACX-MPI, MPI-conforming parallel applications can run on a Computational Grid. It enables the coupling of several MPI applications running as a single virtual machine.

A simple simulation program representing the behaviour of an adaptive CFD code was used to perform the measurements. The program calculates initially 1,000 mesh cells and ends up with 100,000 mesh cells. In the test version without our support module the number of mesh cells passed to Jostle is the same for each PE.

We used 2 x 20 PEs, i.e. 20 nodes, each with two PEs. Heterogeneous compute power has been simulated by assigning extra work to some PEs. For the homogeneous version, this extra work stayed away.

| | heterogeneous | | homogeneous | |
|---|---|---|---|---|
| | with LB support | no LB support | with LB support | no LB support |
| total runtime | 88.25 sec | 260.17 sec | 89.37 sec | 93.28 sec |
| idle times[1] | $< 3$ sec | $< 29$ sec | $< 2.3$ sec | $< 2$ sec |
| ALB Jostle | 3.1 sec | 8.5 sec | 1.7 sec | 8.2 sec |
| LB overhead | 7.5 sec | | 9.48 sec | |

[1] due to blocking send/receives

The results shown in this table expose a high potential of our method. The results of the homogeneous version illustrate the runtime benefit achieved by utilising the hierarchy. Load balancing is mostly done between PEs on the same node (using shared memory communication). This leads to much lower runtimes of jostle compared to the version without support where all PEs are involved in each load balancing step.

The heterogeneous version shows a big gain obtained by our software. The adaptation of load to each PE's capacity shortens the runtime to one-third. The idle waiting times, that decrease performance, can be reduced to nearly the tenth part.

Although the test runs had been done in an environment with very small sized subsystems at lowest hierarchy level (the two PEs, residing on one node), better results are gained compared to runs without our support. More improvements can be expected in Grids with more PEs in low level subsystems. where load can be redistributed between more than two PEs.

Reducing the overhead for load balancing support, e.g. by optimising the code, will lead to further runtime reductions.

## 8. Conclusion

A tool has been presented that supports load balancing for Grid-enabled applications by analysing the environment. Load balancing decisions are based on a hierarchical structure of subsystems that represents different classes of communication speeds. By doing load balancing on a subset of PEs with fast connections, the load balancing overhead is significantly reduced. We showed that adapting the load to the different capacities of the compute nodes, leads to further drastically runtime improvements.

The integration of the presented tool requires only few modifications to the application code. Since we use no global instance, our tool is able to support applications using thousands of PEs.

## References

[1] Infiniband. http://www.infinibandta.org/home.

[2] MPICH homepage. http://www-unix.mcs.anl.gov/mpi/mpich.

[3] PACX-MPI homepage. http://www.hlrs.de/organization/pds/projects/pacx-mpi.

[4] Bruce Hendrickson and Karen Devine. Dynamic load balancing in computational mechanics. In *Computational Methods in Applied Mechanical Engeneering*, volume 184, pages 485 – 500, http://www.cs.sandia.gov/ kddevin/main_publist.html, 2000.

[5] Thilo Kielmann, Henri E. Bal, Jason Maassen, Rob van Nieuwpoort, Lionel Eyraud, Rutger Hofman, and Kees Verstoep. Programming environments for high-performance Grid computing: the Albatross project. *Future Generation Computer Systems*, 18:1113–1125, 2002.

[6] Paul Messina. Distributed supercomputing applications. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[7] Kirk Schloegel, George Karypis, and Vipin Kumar. A unified algorithm for load-balancing adaptive scientific simulations. In *Supercomputing 2000*, 2000.

[8] C. Walshaw and M. Cross. Multilevel mesh partitioning for heterogeneous communication networks. *Future Generation Comput. Syst.*, 17(5):601 – 623, 2001.

[9] Marc H. Willebeek-LeMair and Anthony P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(9):979 – 993, 1993.

[10] Mohammed Javeed Zaki, Wei Li, and Srinivasan Parthasarathy. Customized dynamic load balancing for a network of workstations. *Journal of Parallel and Distributed Computing*, pages 156 –162, 1995.